TP de Programmation Objet en C++ n^{o}1

Mise à niveau en Master 2 Informatique

Septembre 2010

1 Première classe : des `Vecteurs`

- Ecrire une classe `Vecteur` (de l'espace à 3 dimensions) permettant de réaliser les opérations suivantes :
  - création d'un `Vecteur` à l'aide de 3 valeurs,
  - création d'un `Vecteur` par recopie d'un autre `Vecteur`,
  - obtention de chaque composante d'un `Vecteur`,
  - norme d'un `Vecteur`,
  - produit scalaire de deux `Vecteurs`,
  - addition de deux `Vecteurs`,
  - soustraction de deux `Vecteurs`,
  - produit vectoriel de deux `Vecteurs`,
  - détermination de la colinéarité de deux `Vecteurs`,
  - affichage d'un `Vecteur`.

  Il faudra pour cela :
  - → définir les attributs (protégés) permettant de caractériser un `Vecteur` : ses 3 composantes réelles,
  - → définir des fonctions membres (publiques) permettant de réaliser les opérations décrites ci-dessus.
- Ecrire un programme principal permettant de tester l'ensemble des méthodes applicables à un `Vecteur`.

Difficultés abordées

⇒ Création d'une première classe
- → constructeurs et destructeurs
- → fonctions membres publics
- → déclaration d'attributs privés ou protégés

⇒ Premier usage de cette classe

⇒ Complétude du test de cette classe

2 Des `Vecteurs` plus simples à manipuler

- Ajouter à la classe `Vecteur` de nouveaux opérateurs permettant de les manipuler plus facilement :
  - → $+ - * / = == != =$,
  - → $<<$ (injection dans un flot, pour l'affichage).
- Augmenter le programme principal de façon à tester l'ensemble de ces nouvelles fonctions.

Difficultés abordées

⇒ Surcharge des opérateurs

⇒ Factorisation de code à l'aide de méthodes protégées ou privées :
- → entre constructeur par recopie et opérateur d'affectation
- → entre destructeur et opérateur d'affectation

3 Seconde classe : des `Points`

- Ecrire une classe `Point` (de l'espace à 3 dimensions) permettant de réaliser les opérations suivantes :
  - création d'un `Point` à l'aide de 3 valeurs,
  - obtention de chaque composante d'un `Point`,
  - soustraction de deux `Pointss` (cela doit fournir un résultat de type `Vecteur`!),
  - affichage d'un `Point`.

  Il faudra pour cela :
  - → définir les attributs (protégés) permettant de caractériser un `Point` : ses 3 composantes réelles,
  - → définir des fonctions membres (publiques) permettant de réaliser les opérations décrites ci-dessus.
- Augmenter le programme principal de façon à tester l'ensemble des méthodes applicables à un `Point`.
- Ajouter un constructeur à la classe `Vecteur` paramétré par 2 `Points` à relier.

Difficultés abordées

⇒ Référencement croisé de 2 classes

4 Des `Points` plus simples à manipuler

- Ajouter à la classe `Point` de nouveaux opérateurs permettant de les manipuler plus facilement :
  - → $-$ $=$ $==$ $!=$,
  - → $<<$ (injection dans un flot, pour l'affichage).
- Augmenter le programme principal de façon à tester l'ensemble de ces nouvelles fonctions.

Difficultés abordées

⇒ Rien de plus que dans les points précédents !

5 Troisième classe : des `Plans`

- Ecrire une classe `Plan` (de l'espace à 3 dimensions) permettant de réaliser les opérations suivantes :
  - création d'un `Plan` à l'aide de 4 valeurs (les coefficients de l'équation du `Plan`),
  - création d'un `Plan` à l'aide de 3 `Points` non alignés per lesquels passe le `Plan`,
  - création d'un `Plan` à l'aide d'un `Point` du `Plan` et d'un `Vecteur` normal au `Plan`,
  - création d'un `Plan` à l'aide d'un `Point` du `Plan` et de deux `Vecteurs` du `Plan`,
  - savoir si un `Point` appartient au `Plan`,
  - savoir si un `Vecteur` est parallèle au `Plan`,
  - savoir si un `Vecteur` est perpendiculaire au `Plan`,
  - affichage d'un `Plan`.

  Il faudra pour cela :
  - → définir les attributs (protégés) permettant de caractériser un `Plan` : les 4 coefficients de son équation cartésienne,
  - → définir des fonctions membres (publiques) permettant de réaliser les opérations décrites ci-dessus.
- Ajouter à la classe `Plan` de nouveaux opérateurs permettant de les manipuler plus facilement :
  - → $=$ $==$ $!=$,
  - → $<<$ (injection dans un flot, pour l'affichage).
- Augmenter le programme principal de façon à tester l'ensemble des méthodes applicables à un `Plan`.

Difficultés abordées

⇒ Factorisation de code entre les différents constructeurs

6 Remarques

- Pour ce premier TP, l'utilisation de pointeurs est interdite pour le passage de paramètres à des fonctions, procédures ou méthodes.